```
head   1.1;
access;
symbols;
locks; strict;
comment      @# @;


1.1
date    97.01.24.01.09.24;    author airey;   state Exp;
branches;
next    ;


desc
@@


1.1
log
@shader pages
@
text
@XXX - Not complete yet!!!
Name

   SGIX_color_type

Name Strings

   GLX_SGIX_color_type
   GL_SGIX_color_type

Version

   $Date: 1996/10/24 21:58:20 $ $Revision: 1.1 $

Number

   89

Dependencies

   SGIX_fbconfig is required
   SGIX_complex affects the definition of this extension
   EXT_histogram affects the definition of this extension
   SGI_color_table affects the definition of this extension

Overview
```

This extension enhances the fbconfig types to support extended range real and complex data types. Complex data formats include storage for both real and imaginary parts of a color. The ranges of the data types are extended from [0,1] to [-m0.n0,m1.n1], e.g., [-1.5,4.0]. Most OpenGL data paths which clamp colors to [0,1] are modified to clamp colors to [-m0.n0,m1.n1]. These extended range data types are useful for imaging calculations and multipass shading operations that often require signed data and/or complex data formats.

Issues

* two concepts are defined in this extension, color components with extended range and storage for complex color components. I'm not sure if the range aspect should be separated from the complex storage aspect -- they both extend GLX so it seemed like a time saver to group them together

* how does an application specify the color range it needs to represent? should the min/max values be interpreted specially in ChooseConfig or should they be of a more limited usefulness exact match style?

* the spec does not distinguish between old-style [0,1] real color and extend range real color (i.e. no attribute bits). Is this okay?

* float input pixel values and float colors automatically support the extended range. Integer colors are still scaled to [0,1]. This means that GetIntegerv of an extended range color can not represent an extended color range. Is this okay?

* when the color range is extended this implies that accumulation buffers and texture colors also support the extended range. Is this okay?

* if a component isn't stored in the framebuffer (bits == 0) what should range queries return. 0 seems obvious for R, G, and B, but what about alpha. If there is no alpha value, should alpha be the max value?

* do transparent pixel values need to be embellished?

* is a complex ClearColor and ClearAccum needed?

New Procedures and Functions

None

New Tokens

Accepted by the <pname> parameters of GetBooleanv, GetIntegerv,

GetFloatv, and GetDoublev and accepted by the <attrib_list> parameter of glXChooseFBConfigSGIX:

MIN_RED_SGIX	0x????
MAX_RED_SGIX	0x????
MIN_GREEN_SGIX	0x????
MAX_GREEN_SGIX	0x????
MIN_BLUE_SGIX	0x????
MAX_BLUE_SGIX	0x????
MIN_ALPHA_SGIX	0x????
MAX_ALPHA_SGIX	0x????

Accepted by the <attributes> parameter of glXGetFBConfigAttribSGIX, and by the <attrib_list> parameter of glXChooseFBConfigSGIX:

GLX_COLOR_TYPE_SGIX	0x????
GLX_MIN_RED_SGIX	0x????
GLX_MAX_RED_SGIX	0x????
GLX_MIN_GREEN_SGIX	0x????
GLX_MAX_GREEN_SGIX	0x????
GLX_MIN_BLUE_SGIX	0x????
GLX_MAX_BLUE_SGIX	0x????
GLX_MIN_ALPHA_SGIX	0x????
GLX_MAX_ALPHA_SGIX	0x????

Returned by glXGetFBConfigAttribSGIX (when <attribute> is set to GLX_COLOR_TYPE_SGIX) and accepted by the <attrib_list> parameter of glXChooseFBConfigSGIX (following the GLX_COLOR_TYPE_SGIX token):

GLX_REAL_COLOR_SGIX	0x????
GLX_COMPLEX_COLOR_SGIX	0x????

Additions to Chapter 2 of the 1.0 Specification (OpenGL Operation)

Section 2.7 Vertex Specification

Versions of the Color command that take floating-point values accept values nominally between <min> and <max>. <min> corresponds to the minimum value while <max> corresponds to the maximum (machine dependent) value that a compone tmay take on in th framebuffer.

The three component variants of the color command set A to <max>.

Section 2.12 Colors and Coloring

Integer colors are still scaled to the range [0,1] and thus cannot

directly represent colors outside the range [0,1]. Floating-point colors can be used to represent colors outside the range [0,1].

Section 2.12.6 (Clamping or Masking)

After lighting, RGBA colors are clamped to the range [min, max].

Section 2.12.9 (Final Color Processing)

For an RGBA color, each color component (which lies in [min,max]) is converted (by rounding to nearest) to a fixed-point value with m bits for the each of the real and imaginary parts. We assume that the fixed-point representation used represents each value min+(max-min)k/(2^m - 1), where k is in {0, 1, ..., 2^m - 1}, as k (e.g. <max> is represented in binary as a string of all ones). m must be at least as large as the number of bits in the corresponding component of the framebuffer. If the framebuffer is complex, then m must be at least as large as the number of bits in the corresponding real or imaginary part of the component of the framebuffer.

[are the rules for the msbs of unsigned color values matching the framebuffer components true for color ranges outside [0,1]?]

Additions to Chapter 3 of the 1.0 Specification (Rasterization)

Section 3.6.3 (Rasterization of Pixels)

Final Expansion to RGBA

if a group does not contain an A element, then A is added and set to 1.0. If any of R, G, or B is missing from the group, each missing element is assigned a value of 0.0.

RGBA to RGBA Lookup

First, each component is clamped to the range [min,max]. If there is a table associated with each of the R, G, B, and A component elements: PIXEL_MAP_R_TO_R for R, PIXEL_MAP_G_TO_G for G, PIXEL_MAP_B_TO_B for B, and PIXEL_MAP_A_TO_A for A. Each element is multiplied by t/(max-min) where t is an integer one less than the size of the corresponding table, and, for each element, and address is found by resounding this value to the nearest integer. For each element, the addressed value in the corresponding table replaces the element.

Histogram

If HISTOGRAM_EXT is enabled and the width of the table is non-zero, and

the pixel groups contain RGBA values, then indices Ri, Gi, Bi, and Ai are derived from the red, green, blue, and alpha components of each pixel group (without modifying these components) by clamping the components to [min,max], multiplying each by t/(max-min) where t is one less than the width of the histogram table, and rounding each to the nearest integer.

Color Table

The color components of each group that are being replaced by table values are converted to indices by clamping the components to [min,max], multiplying each by one t/(max-min) where t is one less than the width of the color table, and rounding each to the nearest integer. The component value (R, G, B, or A) is then replaced by the value in color table indicated in table E14.2, at the computed index.

Final Conversion

For RGBA components each element is clamped to [min,max]. The resulting values are converted to fixed-point according to the rules given in section 2.12.9 (Final Color Processing).

Section 3.8 (Texturing)

Each R, G, B, and A value so extracted is clamped to [min,max]. Each of the four values set by TEXTURE_BORDER_COLOR is clamped to lie in [min,max].

Section 3.8.3 (Texture Environments and Texture Functions)

TEXTURE_ENV_COLOR is set to an RGBA color by providing four single-precision floating-point values in the range [min,max] (values outside this range are clamped to it).

R, G, B, and A values after being obtained from a supplied texture image, are in the range [min, max].

[modifications to table 3.9???]

Section 3.9 (Fog)

Each component of Cf is clamped to [min, max] when specified.

Additions to Chapter 4 of the 1.0 Specification (Per-Fragment Operations and the Frame buffer)

Color buffers consiste of either unsigned integer color indices or R, G, B, and , optionally A unsigned or signed integer values. The values may also consist of real and imaginary parts.

Section 4.1.3 (Alpha Test)

red is clamped to lie in [min, max], and then converted to a fixed-point value according to the rules given for an A component in section 2.12.9.

Section 4.2.3 (Clearing the Buffers)

void ClearColor(clampf r, clampf g, clampf b, clampf a);

sets the clear value for the color buffers in RGBA mode. Each of the specified components is clapmed to [min, max] and converted to fixed-point according to the rules of section 2.12.9.

void ClearAccum(clampf r, clampf g, clampf b, clampf a);

takes four floating-point arguments that are the values, in order, to which to set the R, G, B, and A values of the accumulation buffer (see the next section). These values are clamped to the range [min2,max] where min2 = minimum(-1,<min>) when they are specified.

Section 4.2.4 (The Accumulation Buffer)

Accumulation buffer values are taken to be signed values in the range [min2,max] where min2 = minimum(-1,<min>). Using ACCUM obtains the R, G, B, and A components from the buffer current selected for reading (section 4.3.2). Each component, considered as a fixed-point values in [min,max] (see section 2.12.9) is converted to floating-point. Each result is then multiplied by value.

The color components operated on by Accum must be clamped only if the operation is RETURN. In this case, a value sent to the enabled color buffers is first clamped to [min,max]. Otherwise, results are undefined if the result of an operation on a color component is too large (in magnitude) to be represented by the number of available bits.

4.3.2 (Reading Pixels)

Conversion to RGBA values

The R, G, and B (and possibly A) values form a group of elements. Each element is taken to be a fixed-point value in [min,max] with m bits, where m is the number of bits in the corresponding color component of the selected buffer (see section 2.12.9)

Final Conversion

For a component, if the type is FLOAT then each component is first clamped to [min, max]; if the type is not FLOAT then each component is first clamped to [0,1]. Then the appropriate conversion formula from Table 4.7 is applied to the component.

Additions to Chapter 5 of the 1.0 Specification (Special Functions)

None

Additions to Chapter 6 of the 1.0 Specification (State and State Requests)

None

Additions to the GLX Specification

[Added to the description of glXChooseFBConfigSGIX]

If any of GLX_MIN_RED_SGIX, GLX_MAX_RED_SGIX, GLX_MIN_GREEN_SGIX, GLX_MAX_GREEN_SGIX, GLX_MIN_BLUE_SGIX, GLX_MAX_BLUE_SGIX, GLX_MIN_ALPHA_SGIX, or GLX_MAX_ALPHA_SGIX are specified in <attrib_list> then the value that follows indicates the minimum or maximum value which can be represented in the OpenGL pipeline as a color. Any OpenGL state which stores color values (e.g. current color, texture data, the color buffer, accumulation buffer, aux buffers, etc) can represent values between the minimum and maximum range. Values are specified as integral numerator, denominator pairs (rational numbers).

If GLX_COLOR_TYPE_SGIX is in <attrib_list> then the value that follows indicates how color values are represented in the framebuffer:

GLX_REAL_COLOR_SGIX specifies that only the real value of each color component is stored in the color and accumulation buffers. The actual representation of the data is unspecified, but the range that can be represented can be queried using the color component min and max requests.

GLX_COMPLEX_COLOR_SGIX specifies that each color component is stored as a complex number. The distribution of bits used to represent the real and imaginary portions of the complex number are unspecified as is the actual representation of the data (two's complement, sign magnitude, floating point etc). GLX_RED_SIZE, GLX_GREEN_SIZE, GLX_BLUE_SIZE, GLX_ALPHA_SIZE are equal to the sum of the number of framebuffer bits used to store both the real and imaginary parts.

If an accumulation buffer is present, then it also stores both real and imaginary parts for each color component present. GLX_ACCUM_RED_SIZE, GLX_ACCUM_GREEN_SIZE, GLX_ACCUM_BLUE_SIZE, and GLX_ACCUM_ALPHA_SIZE are equal to the total number of bits used to store each complex component.

In general, only non-displayable drawable types (e.g. GLXPbuffers) will be able to be created with FBConfigs which support extended range real or complex color component types (see GLX_DRAWABLE_TYPE_SGIX).

GLX Protocol

[tbd]

Dependencies on EXT_histogram

If EXT_histogram is not implemented, then the references to GetHistogramEXT and GetMinmaxEXT in this file are invalid, and should be ignored.

Dependencies on SGI_color_table

If SGI_color_table is not implemented, then the references to ColorTableSGI and GetColorTableSGI in this file are invalid, and should be ignored.

Dependencies on SGIX_complex

If SGIX_complex is not supported, references to GLX_REAL_COLOR_SGIX, GLX_COMPLEX_COLOR_SGIX and GLX_COLOR_TYPE_SGIX in this document are invalid and should be ignored.

Errors

None

New State

None

New Implementation Dependent State

None

@